Reference 1: JP 7-84765 A

[0020] Moreover, in the above embodiment of the present invention timing with which an event is sent is not taken into consideration to simplify the explanations. By locating an event synchronization section, however, an event can also be sent with timing taken into consideration. This is realized in the following way. An event structure of the X caused by an input device includes event occurrence time. If an event is generated from event information saved by the event record section (11a) and is sent, the event synchronization section for obtaining the differential between events from event occurrence time recorded and for sending the events with actual timing is located. By doing so, automatic operation can be performed smoothly. In addition, synchronous speed can easily be increased or decreased by changing timing calculation.

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 07-084765
(43)Date of publication of application : 31.03.1995

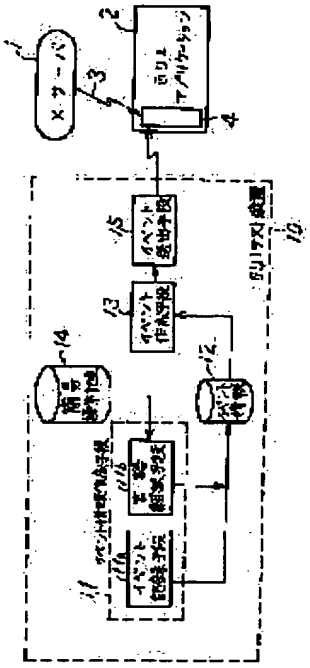
(51)Int.Cl. G06F 9/06
G06F 9/45
G06F 11/28
G06F 15/00

(21)Application number : 05-195669
(22)Date of filing : 06.08.1993

(71)Applicant : TOSHIBA CORP
(72)Inventor : MATSUOKA YUICHIRO

(54) GUI TEST DEVICE

(57)Abstract:
PURPOSE: To reduce the handling time and period required for a GUI application test by enabling the GUT application test even if a mouse operation is not actually performed by describing the test contents by a simplified program language.
CONSTITUTION: An event information preparation means 11 prepares the event information 12 corresponding to a GUI(graphical user interface) operation for a test. An event record means 11a extracts event information from an actual GUI operation, a language translation means 11b converts a simple operation description into event information 12, and the information is recorded and preserved in a file. An event preparation means 13 prepares event data which can be actually transmitted to a GUI application 2 from the event information 12 prepared/preserved by the means 11. An event transmission means 15 transmits the prepared event to the application to be a test object. When the application receives this event, the application operates as if an input were actually performed from a mouse, etc.



LEGAL STATUS

[Date of request for examination] 07.09.1999
[Date of sending the examiner's decision of rejection] 08.01.2002
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平7－84765

(43)公開日　平成７年(1995)３月31日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ	9/06		9367－5B		
	9/45				
	11/28	３４０　Ａ	9290－5B		
	15/00		7459－5L		

審査請求　未請求　請求項の数３　ＯＬ　（全 8 頁）

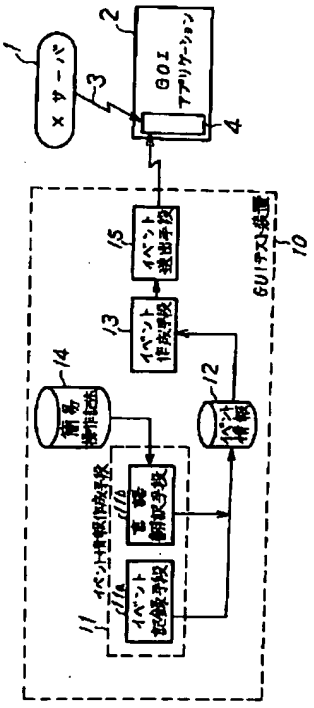
(21)出願番号　特願平5－195669

(22)出願日　平成５年(1993)８月６日

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地
(72)発明者　松岡　雄一郎
神奈川県川崎市幸区小向東芝町１番地　株式会社東芝研究開発センター内
(74)代理人　弁理士　則近　憲佑

(54)【発明の名称】　ＧＵＩテスト装置

(57)【要約】
【目的】本発明は、ＧＵＩアプリケーション開発における修正とテストの繰り返し作業の際に、繰り返し行われるＧＵＩの一連の操作を保存・再利用し、煩わしいＧＵＩの操作を自動的に行い、開発者は操作後の結果を確認するだけでよいＧＵＩテスト装置を提供することにある。
【構成】本発明では、ＧＵＩのテスト時に行うＧＵＩの操作に対応するイベントの情報を作成し、テストの際にこのイベント情報からイベントを作成・送出することにより、ＧＵＩの操作を自動的に行う。
【効果】本発明により、従来のＧＵＩテストの際に煩わしかった操作を自動化でき、ＧＵＩ開発者の負担を削減することができる。



1

【特許請求の範囲】

【請求項1】ＧＵＩを用いたアプリケーションのテストにおいて、アプリケーションのテスト対象動作を引き起こすＧＵＩの操作に対応するイベントの情報を作成するイベント情報作成手段と、前記イベント情報作成手段によって作成されるイベントの情報から実際のイベントを作成するイベント作成手段と、前記イベント作成手段によって作成されたイベントをテスト対象のアプリケーションに送出するイベント送出手段とを設けたことを特徴とするＧＵＩテスト装置。

【請求項2】イベント情報を作成するイベント情報作成手段は、入力デバイスの操作等によってアプリケーションに発生した実際のイベントからイベント情報を抽出して記録するイベント記録手段を設けたことを特徴とする請求項1記載のＧＵＩテスト装置。

【請求項3】イベント情報作成手段は、簡易言語による記述をイベント情報に変換する言語翻訳手段を設けたことを特徴とする請求項1記載のＧＵＩテスト装置。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、ワークステーション等におけるウィンドウシステム上のＧＵＩ（グラフィカル・ユーザー・インターフェース）アプリケーションのテスト装置に関する。

【０００２】

【従来の技術】ウィンドウシステムを用いたコンピュータの利用形態が、ワークステーションではもちろん、パーソナルコンピュータにおいても一般的になり、ＧＵＩ（グラフィカル・ユーザー・インターフェース）を持つアプリケーションが広く用いられるようになってきている。ＧＵＩはその名が示す通り、グラフィカルな画面と、また入力手段としてマウスを利用できるため、利用者は直感的にＧＵＩを操作することが可能である。しかしながら、ＧＵＩアプリケーション開発においては、動作テストの際に、各テスト項目の動作を確認するための操作を開発者が直接マウスやキーボードを用いて一々操作する必要がある。また、テストによりバグを見つかりＧＵＩアプリケーションの修正を行ったり、ＧＵＩアプリケーションの機能を一部拡張したりした際には、その修正・拡張の影響が他の部分に及んでいないかを調べるために、全てのテスト項目をテストしなおさなければならず、ＧＵＩアプリケーションのテスト作業には多くの手間と期間とを要するという問題があった。さらに、ＧＵＩ自体がますます複雑化する傾向にあり、ボタンの数やメニューの階層などが増加しているため、テスト項目の増加、各テスト項目のテストシーケンスが長くなることによる操作の誤りの増加、多くのメニュー項目からテスト項目に対応したメニュー項目を探す手間の増大等により、上記の問題がさらに大きなものになってきている。このため、ＧＵＩアプリケーションの開発工程においては、テスト作業がボトルネックとなり、開発期間の長期化が深刻な問題となってきている。

2

【０００３】

【発明が解決しようとする課題】以上のように、従来、ＧＵＩアプリケーションを開発する際には、開発者はアプリケーションの動作を調べるためのテスト工程に、非常に多くの手間と期間とを要するという問題がある。

【０００４】この発明は上記の問題に鑑みてなされたものであり、ＧＵＩアプリケーションのテストの手間と期間とを大幅に低減するＧＵＩテスト装置を提供することを目的としている。

【０００５】

【課題を解決するための手段】この発明は上記の目的を達成するため、ＧＵＩの操作に対応するイベント情報を作成するイベント情報作成手段、即ち、実際のＧＵＩ操作からイベント情報を作成するイベント記録手段と、操作記述を翻訳してイベント情報を作成する言語翻訳手段と、イベント情報からイベントを作成するイベント作成手段と、作成されたイベントをテスト対象のＧＵＩに送出するイベント送出手段を設けたものである。

【０００６】

【作用】この発明によるＧＵＩテスト装置では、イベント情報作成手段がテストのためのＧＵＩ操作に対応するイベント情報を作成する。イベント記録手段は実際のＧＵＩ操作からイベント情報を抽出する。言語翻訳手段は簡易操作記述をイベント情報に変換する。イベント情報は再利用できるように、ファイルに記録され保存される。イベント作成手段はイベント情報作成手段によって作成・保存されたイベント情報から実際にＧＵＩアプリケーションに送ることができるイベントデータを作成し、イベント送出手段は作成されたイベントをテスト対象のアプリケーションに送出する。アプリケーションはこのイベントを受け取ると、実際にマウスやキーボードから入力が行われたかのように動作する。

【０００７】

【実施例】次に、本発明の一実施例を示す。この実施例では、具体化のためにＸウィンドウシステム（以下Ｘと呼ぶ：Ｘ Ｗｉｎｄｏｗ Ｓｙｓｔｅｍは米国Ｍａｓｓａｃｈｕｓｅｔｔｕｓ Ｉｎｓｔｉｔｕｔｅｏｆ Ｔｅｃｈｎｏｌｏｇｙの登録商標）上におけるツールキットを用いたＧＵＩ（グラフィカル・ユーザー・インターフェイス）アプリケーションのテストを行う場合を例にとり説明する。

【０００８】実施例の詳細な説明に入る前に、Ｘにおけるサーバとアプリケーションの関係について図1を利用して簡単に説明する。Ｘサーバ（1）はマウスやキーボード等の入力デバイスを監視し、これらのデバイスから入力が行われると、対応するイベントを作成し、適切なアプリケーション（2）に対して作成したイベントを送出する（3）。このイベントは、各アプリケ

3

ーションが持つイベントキュー（4）に置かれる。アプリケーションはイベントキューからイベントを1つ取り出し、イベントを処理するというループを実行している。アプリケーションはイベントに対する処理を保持しており、受け取ったイベントに対応する処理を実行する。

【0009】次に、本発明の実施例を図面に従って説明する。図1は本発明によるGUIテスト装置（10）の構成及び、テスト対象のアプリケーション、Xサーバとの関係を示している。本テスト装置中のイベント情報作成手段（11）はテストの際に自動化するGUIの操作に対応するイベントの情報（12）を作成する。このイベント情報作成手段（11）としては、実際のGUI操作からイベント情報（12）を作成するイベント記録手段（11a）と簡易言語による記述（14）を翻訳してイベント情報（12）を作成する言語翻訳手段（11b）がある。イベント作成手段（13）はイベント情報作成手段（11）によって作成されたイベント情報（12）からイベントデータを作成する。イベント送出手段（15）は作成されたイベントデータをテスト対象のアプリケーション（2）に送出する。アプリケーション（2）はこのイベントを受け取ると、実際にマウスやキーボードから入力が行われたかのように動作する。イベント列が続けて送出されると、アプリケーション（2）はテスト対象の動作を引き起こすためのGUI操作を自動的に行う。

【0010】上記の構成では、各手段はアプリケーションと独立なものとして説明したが、以下で、イベント記録手段（11a）はアプリケーション（2）の一部として実現する例を示す。

【0011】また、話を簡単にするために、いくつかの前提を設ける。まず、テスト装置からイベントが送られている間も、入力デバイスからの入力は他のアプリケーションに対しても行われないものとする。

【0012】図2はイベント記録手段（11a）がイベント情報（12）を作成する際のフローチャートである。本実施例ではイベント記録手段（11a）はアプリケーション（2）の一部であり、このフローチャートはアプリケーション（2）のイベント処理ループのフローを示している。ここでは、アプリケーション（2）内にイベント情報（12）を記録するかしないかを制御するためのフラグを設けて、フラグが1の間、アプリケーション（2）内のイベント処理ループ中でイベント情報（12）を記録するという方法を用いている。即ち、アプリケーション（2）の起動直後フラグは“0”であり、アプリケーション（2）は通常のイベント処理ループを実行する（21、22、24）。フラグが“1”になると、この時点から発生するイベントはイベント情報（12）としてファイルに記録される（21、22、23、24）。フラグを“0”に戻すと、アプリケーション（2）はイベント情報（12）の記録をやめ、再び通常のイベント処理ループを実行する。このフラグと記録するファイルの名前は外部から制御する。これはXによるクライアント間のメッセージ通信により実現できる。クライアント間メッセージ通信とは、アプリケーションが別のアプリケーションに対してイベントを送ることにより、データやタイミングを知らせるアプリケーション間の通信手段であり、この機能を用いてフラグのセット及びファイル名を外部からアプリケーションに対して通知することができる。このようにしてイベント記録手段（11a）はテスト対象のにGUI（2）に発生したイベントを捕らえ、イベント情報（12）としてファイルに記録する。

4

【0013】このイベント記録手段（11a）はアプリケーション（2）中のイベントキュー（4）を利用してイベント（3）を捕獲するため、アプリケーション（2）の一部を修正する必要がある。具体的には、テスト用のヘッダファイルを加え、イベント処理ループにXtMainLoop（ ）を用いている場合はライブラリを追加してコンパイルすることにより実現できる。イベント処理ループがユーザ作成のものである時には、イベント処理ループの一部にフラグが1の時にイベント情報をファイルに書き出すように修正することによって実現することができる。図3にイベント処理ループのプログラム例の一部を表すプログラム（20）を示す。

【0014】図4は簡単なGUIの例であり、このアプリケーション（41）は3つのボタンA、ボタンB、ボタンCと1つのテキスト入力エリアDから構成されている。図5は図4のアプリケーション（41）に対する自動操作の簡易言語による記述の例である。この内容は、まずアプリケーション（41）のボタンAをマウスでクリックし（51）、テキストエリアDにexampleとキー入力し（52）、ボタンBをマウスクリックする（53）操作を意味している。この操作言語は言語翻訳手段（11b）により、イベント情報（12）に変換される。この変換はCプリプロセッサの機能を用いて実現している。ただし、ウィンドウ等の各種IDは固定のものと仮定している。簡易言語の各操作は、Cプリプロセッサの#defineを用いたマクロとして定義されており、このマクロが定義されたファイルを#includeによって取り込んだ記述言語を、Cプリプロセッサで処理することにより、簡単言語をイベント情報に変換することができる。図6に定義マクロの一例を表わすプログラム（21）を示す。このマクロ定義により、GUI上のボタンを押すというマクロは、ボタンの位置でマウスのボタンをクリックする、即ちマウスボタンを押す、そして離すという2つのイベント情報に展開される。また、メニューの3番目の項目を選択するというマクロは、メニューを表示するためのマウスボタンのクリック、メニュー中の3番目の項目を選択するためのマウ

スボタンのクリックというイベント情報に展開され、対応するイベント情報を生成する。

【0015】次に、上記で説明した2種のイベント情報作成手段（11a、11b）が作成するイベント情報（12）と、このイベント情報（12）からイベント作成手段と情報（12）は上記2種のイベント情報作成手段（11）により作成される。この実施例では1行が1個のイベントに対応するイベント情報となっている。1行のイベント情報の内容はXのイベント構造体の全メンバの値であり、イベントの型によって異なるが、イベントの型（ＢｕｔｔｏｎＰｒｅｓｓ、ＫｅｙＰｒｅｓｓ等）、ディスプレイＩＤ、イベントが発生したウィンドウのＩＤ、ルートウィンドウＩＤ、イベント発生時刻、イベント発生時のマウスカーソルの場所（x、y座標）等を持っている。図8にイベントデータ（22）の例を示す。これはXのイベント構造体のうち、マウスボタンのＰｒｅｓｓ・Ｒｅｌｅａｓｅ時に発生するイベントのデータ構造である。図7のイベント情報（23）はマウスボタンのＰｒｅｓｓ・Ｒｅｌｅａｓｅの際の具体例であり、このイベント情報がイベント作成手段（13）によって第8図に示すようなイベントデータに変換される。また、この実施例では、一度の自動操作に対応するイベント情報は一つのファイルに格納される。即ち、ある自動操作を行う場合、そのイベント情報を持つファイルを指定することにする。

【0016】次に、イベント作成手段（13）とイベント送出手段（15）により、アプリケーションのテスト対象動作が起動するまでのＧＵＩ操作を自動的に実行する方法について述べる。

【0017】図9はＧＵＩの操作を自動実行する際のフローチャートである。前提として、自動実行する操作のイベント情報がイベント情報作成手段（11a、11b）によって作成されているものとする。以下では、イベント作成手段（13）が実行する処理フローを説明する。まず、自動実行する操作に対応するイベント情報が記録されたファイルを指定する（91）。次に、指定ファイルから1行、即ち1イベント情報を読み込む（92）。このイベント情報からイベント構造体の各メンバの値を獲得し、イベントデータを作成する（93）。作成されたイベントはイベント送出手段（15）がアプリケーションに送出する（94）。この後、ファイルの終わりまで再びイベント情報の読み出しからの処理を繰り返し、ファイル内のイベント情報を全てイベントとして送出した後、終了する。

【0018】上記実施例では、テスト装置からイベントが送られている間は、実際のマウスやキーボードからの入力は他のアプリケーションに対しても行われないものと仮定したテスト装置を説明してきたが、テスト装置とXサーバの双方からのイベントを監視し、テスト装置からイベントが送られてくる間、Xサーバからのイベントを制御するイベント監視手段を設けることにより、テスト装置の起動中でも入力デバイスを扱えるようにすることもできる。このイベント監視手段とアプリケーションとの関係を図10に示す。このイベント監視手段（103）を実現するために、イベント中のメンバにフラグを持たせ、テスト装置（101）から送るイベントのフラグにXサーバ（102）からのイベントと異なる値を持たせることにより、双方からのイベントが区別できるようになる。イベント監視手段（103）はこのフラグによって双方のイベントを見分け、Xサーバ（102）からのイベントを制御する。例えば、テスト装置の起動中、サーバからのマウスやキーボードに関するイベントは監視手段（103）で拒否することによりアプリケーションは入力デバイスから独立し、テスト中でも他のアプリケーションに対して入力デバイスを扱うことができる。

【0019】また、上記実施例では、イベント記録手段（11a）がアプリケーション（2）内でイベントを捕らえることにより、イベント情報（11）を記録する例を示したが、イベントの捕らえる場所をアプリケーション（2）ではなく、Xサーバ（1）内、またはXサーボ（1）に近いところで捕らえることにより、アプリケーション（2）に手を加えずに、上記の動作を実施するようにすることができる。これは、Ｘ１１ｂの関数を用いてXサーバ（1）とイベントに関する情報交換を行うこと、また必要ならばXサーバ（1）を拡張することにより実現できる。

【0020】また、上記実施例では、説明を簡単にするためイベント送出のタイミングについては考えずに説明してきたが、イベント、同期手段を設けることにより、タイミングを考慮したイベント送出も行うことができる。これは以下のように実現する。入力デバイスによって発生するXのイベント構造体はイベントの発生時刻をもつ。イベント記録手段（11a）によって保存されるイベント情報からイベントを作成・送出する場合、記録されたイベント発生時刻からイベント間の差分を得て、実際に行われたタイミングでイベントを送出するイベント同期手段を設けることにより、自動操作を円滑に行うことができる。また、同期速度を速めたり、遅くしたりすることも、タイミングの計算を変更することにより、容易に実現できる。

【0021】また、上記実施例では、イベントの送出は連続的に行われるが、対話型のイベント送出手段（15）を設けることにより、イベントを一つ一つ対話的に送ることもできる。これは、イベント作成手段（13）がイベントを作成した後、プロンプトを出力し、ユーザからの入力を待つ。入力が行われるとイベントを送出することにより実現できる。この手段により、イベントの速度を制御するとともに、エラーが起こった場合にどのイベントでエラーが起こったかを容易に知ることができ

7

るようになる。また、コマンドとして、ＤＢＸのような機能、例えば送出イベントの値をセットできたり、イベントのトレースを行うような機能を加えることにより、送出されるイベントの詳細情報を得ながら、対話的にイベントを送出することが可能となる。

【００２２】

【発明の効果】以上の説明から明らかなように、本発明によれば、簡易言語によりテスト内容を記述しておけば、実際にマウス操作を行わなくてもＧＵＩアプリケーションのテストができ、ＧＵＩアプリケーションのテストに要する手間と期間を大幅に低減することができる。

【００２３】また、実際にマウス操作を行ってＧＵＩアプリケーションのテストを行った場合でも、テストの情報が記録されるので、ＧＵＩアプリケーションのデバッグ・修正後に再度テストする際には、修正前にテストした項目については、自動的に再テストすることができ、ＧＵＩアプリケーションのテスト作業の効率を大幅に向上させることができる。

【図面の簡単な説明】

【図１】　本発明の構成図。

【図２】　実際の操作からイベント情報を記録する際の、アプリケーションのイベント実行ループのフローチャート。

8

【図３】　図５で示すフローを実現したプログラムの一部を示す図。

【図４】　簡単なＧＵＩアプリケーションを示す図。

【図５】　簡易言語により記述の例を示す図。

【図６】　簡易言語のためのマクロの例を示す図。

【図７】　イベント情報を示す図。

【図８】　Ｘのマウスボタンの Ｐｒｅｓｓ・Ｒｅｌｅａｓｅに関するイベント構造体を示す図。

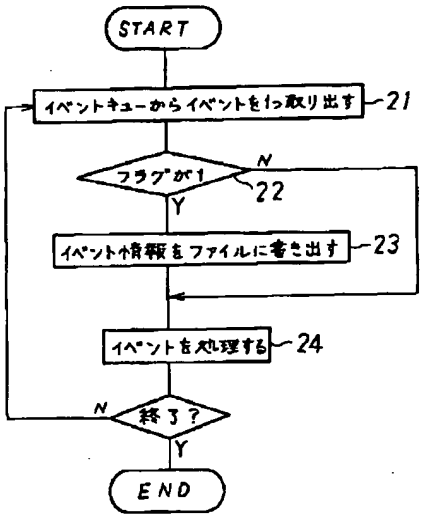
【図９】　テスト時の自動操作を実現する処理を示すフローチャート。

【図１０】　イベント監視手段とアプリケーションとの関係を示す図。

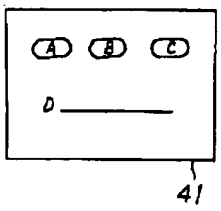
【符号の説明】

１…Ｘサーバ、　２…ＧＵＩアプリケーション、　３…イベント、４…イベントキュー、　１１…イベント情報作成手段、１１ａ…イベント記録手段、　１１ｂ…言語翻訳手段、　１２…イベント情報、１３…イベント作成手段、　１４…簡易操作言語、　１５…イベント送出手段、４１…ＧＵＩアプリケーション、　５１～５３…簡易操作言語、　１０１…イベント送出手段、　１０２…Ｘサーバ、　１－３イベント監視手段、１０４…ＧＵＩアプリケーション。

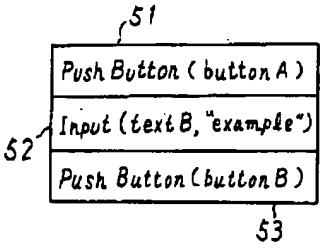
【図２】



【図４】



【図５】

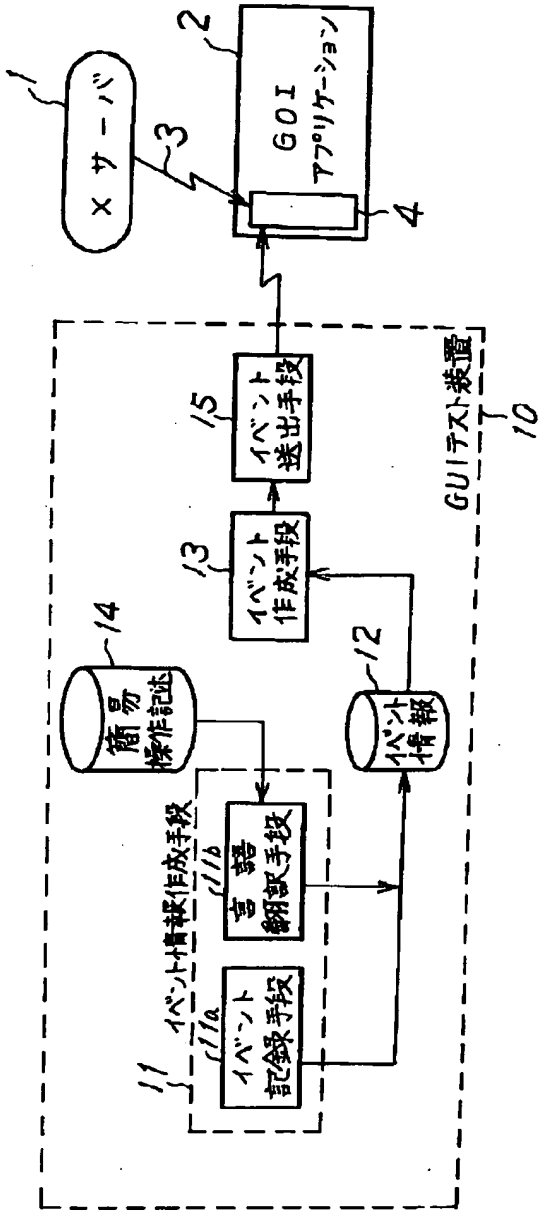


【図８】

```
typedef struct {
        int type;
        unsigned long serial;
        Bool send_event;
        Display *display;
        Window window;
        Window root;
        Window subwindow;
        Time time;
        int x, y;
        int x_root, y_root;
        unsigned int state;
        unsigned int button;
        Bool same_screen;
} XButtonEvent;
```

22

【図1】

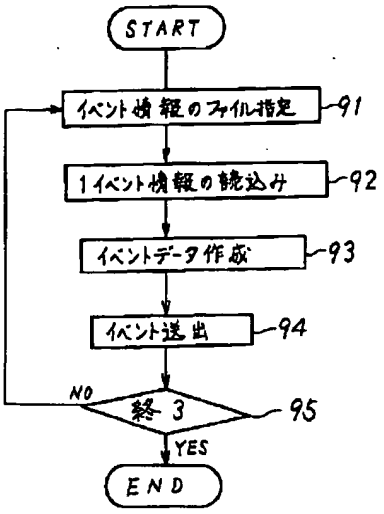

1
×サーバ
2
GUI
アプリケーション
3
4
GUIテスト装置
10
15
イベント
送出手段
13
イベント
作成手段
14
簡易
操作記述
12
イベント
情報
イベント情報作成手段
11
11b
言語
翻訳手段
11a
イベント
記録手段

【図3】

```
#define XtMainLoop  main_roop

int     STORE_MODE = FALSE;
void main_loop()
{
  XEvent      event;

  while(1) {
        XtNextEvent(&event);

        if(STORE_MODE == TRUE)
                    store_event((event,  file);
          XtDispatchEvent(&event);

  }

  }


  void store_event(event, file)
  XEvent event;
  FILE   *file;
  {
     switch(event.type)
      {
     /** XButtonEvent **/
     case ButtonPress:
     case ButtonRelease:
      fprintf(file,"%d %lu %d %d %lu %lu %lu %lu %d %d %d %d %d %ld %ld
                                  event.xbutton.  type,
                                  event.xbutton.serial,
                                  event.xbutton.send_event,
                                  (int)event.xbutton.display,
                                  event.xbutton.window,
                                  event.xbutton.root,
                                  event.xbutton.subwindow,
                                  event.xbutton.time,
                                  event.xbutton.x,
                                  event.xbutton.y,
                                  event.xbutton.x_root,
                                  event.xbutton.y_root,
                                  event.xbutton.state,
                                  event.xbutton.button,
                                  event.xbutton.same_screen);
              break;
              /** XKeyEvent **/
              case KeyPress:
              case KeyRelease:
                        .
                        .
                        .
               } /** end switch **/

  }
```

20

【図9】

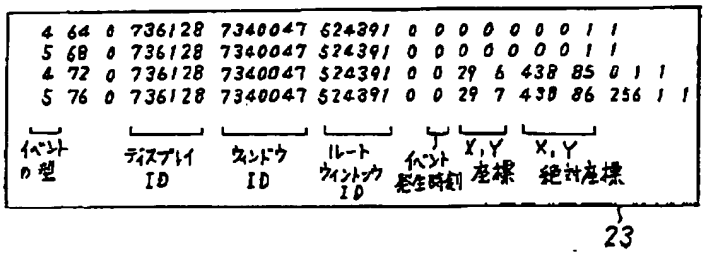


【図6】

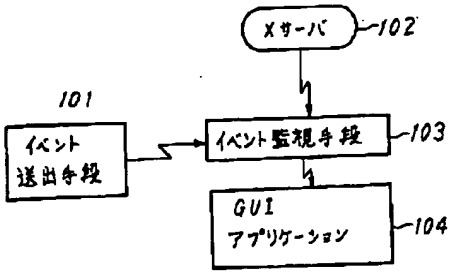
```
#define DISPLAY 736128
#define ROOTWIN 524391
#define buttonA 7340046

#define PushButton(X) 400 DISPLAY x ROOTWIN 0 0 0 0 0 0 0 1 1¥$0 0 01
SPLAY x ROOTWIN 0 0 0 0 0 0 0 1 1
```

21

【図7】


4 64 0 736128 7340047 624391 0 0 0 0 0 0 0 1 1
5 68 0 736128 7340047 524391 0 0 0 0 0 0 0 1 1
4 72 0 736128 7340047 524391 0 0 29 6 438 85 0 1 1
5 76 0 736128 7340047 524391 0 0 29 7 438 86 256 1 1
イベントの型
ディスプレイID
ウィンドウID
ルートウィンドウID
イベント発生時刻
X,Y座標
X,Y絶対座標
23

【図10】


Xサーバ
102
101
イベント送出手段
イベント監視手段
103
GUIアプリケーション
104